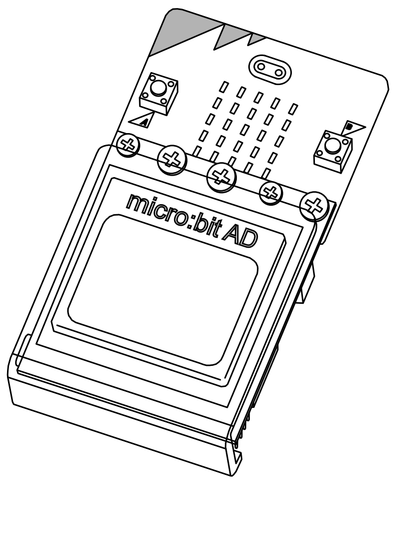
Python programming

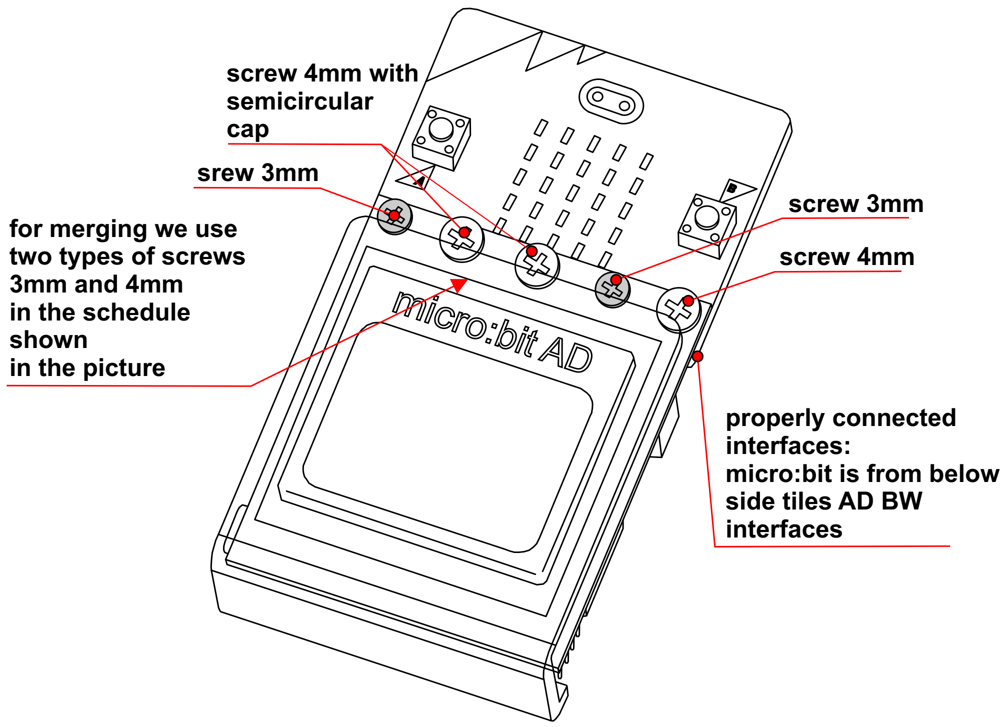
micro:bit AD BW



DIDACTA ADVANCE d.o.o.
Čakovec, Croatia (EU)
www.didacta.hr

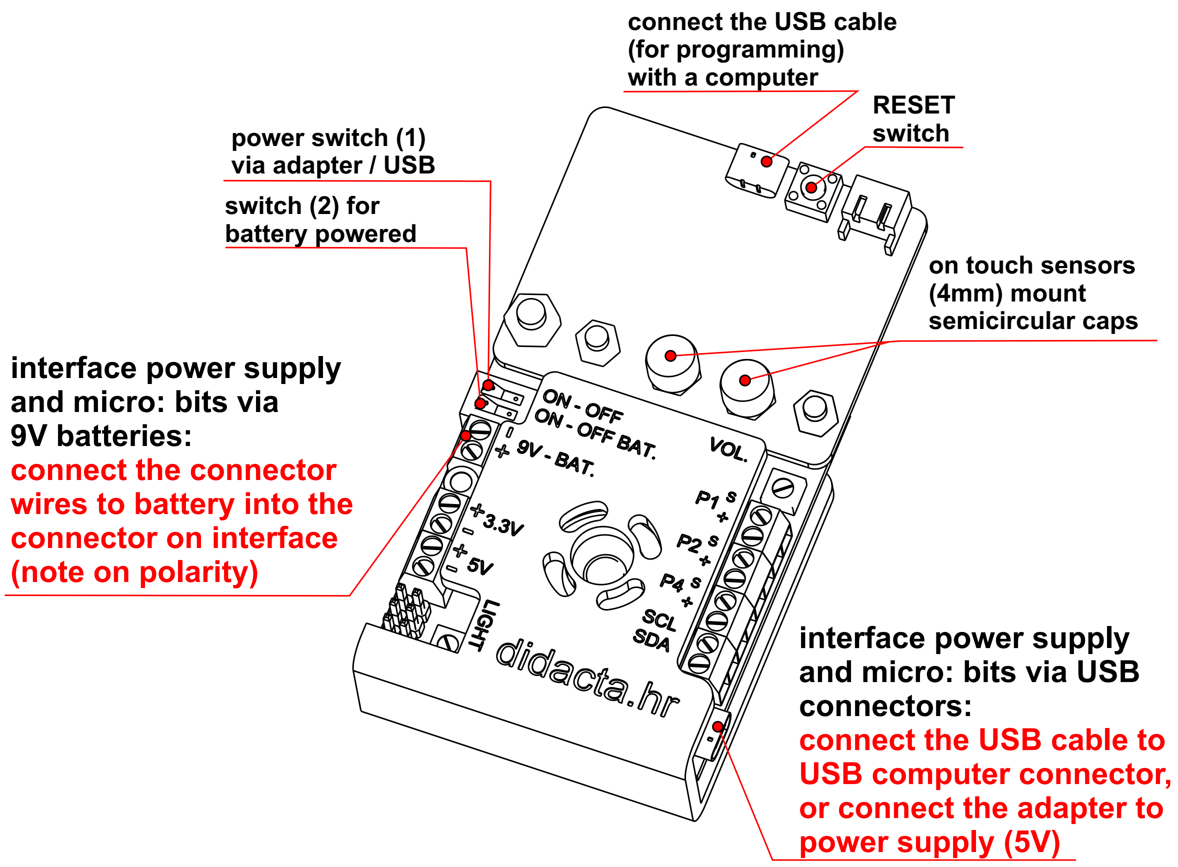
1. BEGINNING

1.1. Connecting micro:bit and AD interface with screws


screw 4mm with semicircular cap
srew 3mm
for merging we use two types of screws 3mm and 4mm in the schedule shown in the picture
screw 3mm
screw 4mm
properly connected interfaces: micro:bit is from below side tiles AD BW interfaces
micro:bit AD

We use smaller screws in positions where damage could occur elements on the micro: bit board, which are located near the connection opening.

1.2. Power supply of micro:bit and AD interface via battery, adapter or USB port


connect the USB cable (for programming) with a computer
RESET switch
power switch (1) via adapter / USB
switch (2) for battery powered
on touch sensors (4mm) mount semicircular caps
ON - OFF
ON - OFF BAT.
9V - BAT.
VOL.
3.3V
5V
P1
P2
P4
SCL
SDA
LIGHT
didacta.hr

interface power supply and micro: bits via 9V batteries:
connect the connector wires to battery into the connector on interface (note on polarity)

interface power supply and micro: bits via USB connectors:
connect the USB cable to USB computer connector, or connect the adapter to power supply (5V)

1.3. Interface launch

After startup, the text shown in Figure 1 will be displayed on the micro:bit AD BW interface screen. The interface is ready to work.

If the program is already loaded in micro:bit, you need to press the RESET button on the micro:bit, to start program.

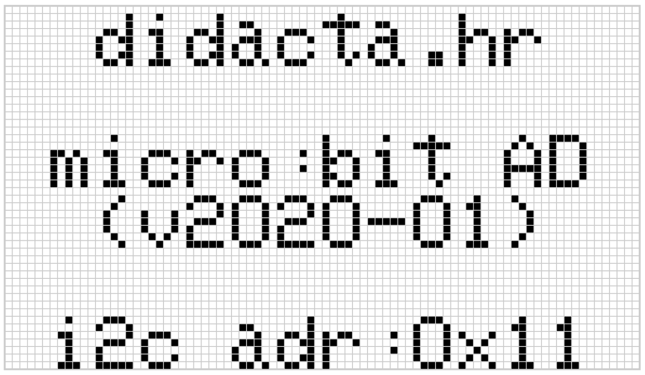


Figure 1.

1.4. Interface screen

The micro: bit AD BW interface has a black and white screen with a graphic resolution of 48 x 84 pixels (Figure 2).

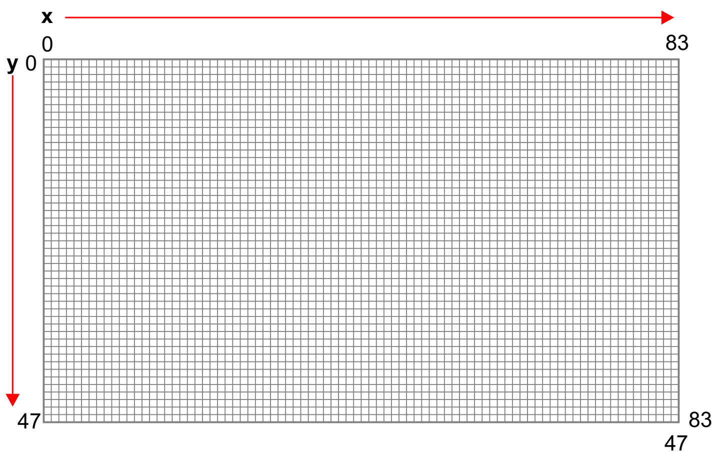


Figure 2. Graphics mode - resolution 48 x 84.

A resolution corresponding to the font size is used to print the text (text mode). Standard size text character (font font is 7 x 5 pixels) is 8 x 6 pixels with space pixels. That is why it is a resolution to print 6 x 14 characters (Figure 3). When creating a program, we must take into account which program commands we also use which mode of operation they are intended for.

Graphic functions use graphic resolution (line, circle, rectangle, ...), and text mode is used in standard text printing (not graphic) and when defining the game screen, and positioning the player object.

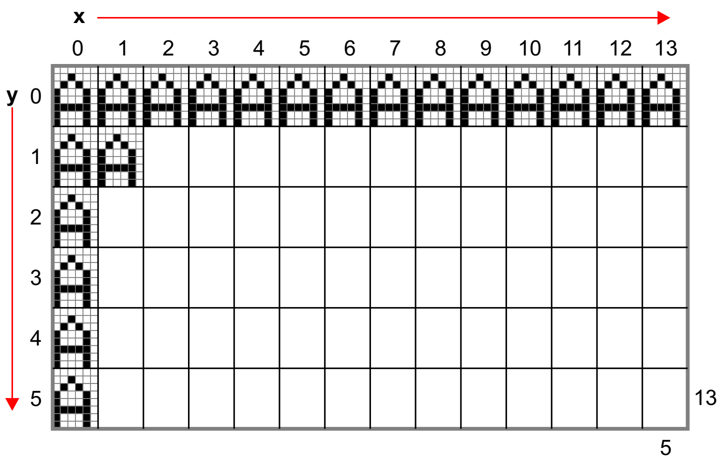


Figure 3. Text mode - 6 x 14 resolution.

2. MicroPython (MU)

2.1. MicroPython

You can download MicroPython (MU) via the link below and install it on your computer:

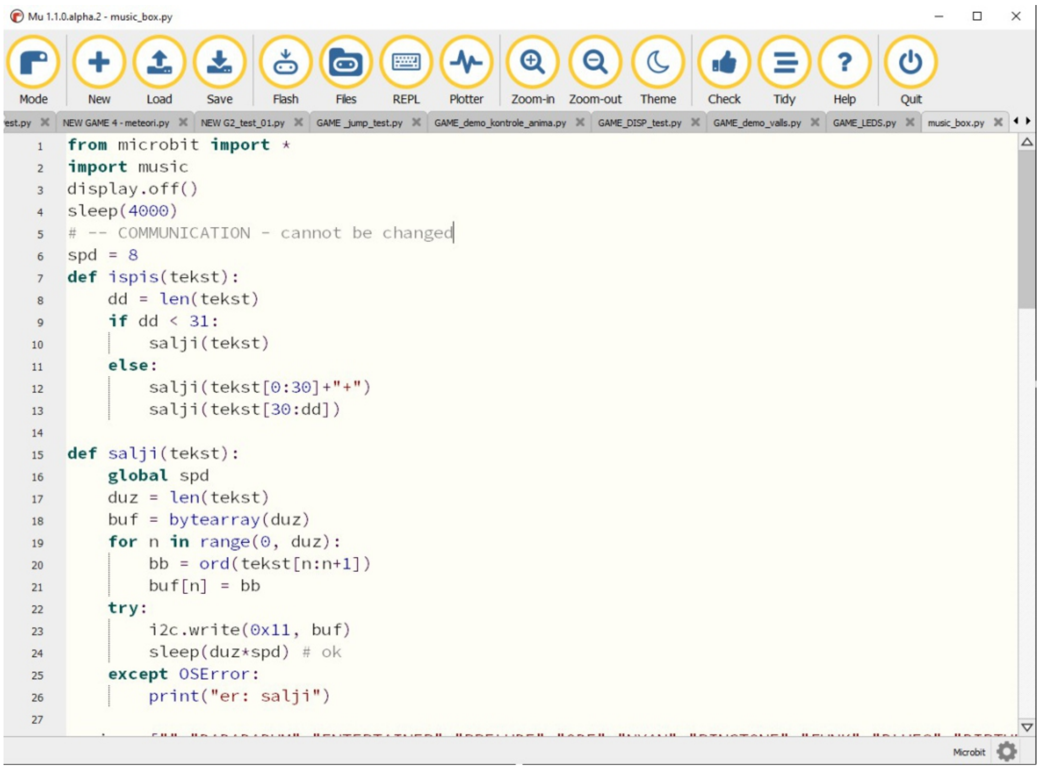
https://codewith.mu/en/download

If you do not want to install the program on your computer, you can use the version that works in a web browser:

https://python.microbit.org/v/2

Programming instructions can be found at:

https://microbit.org/get-started/user-guide/python

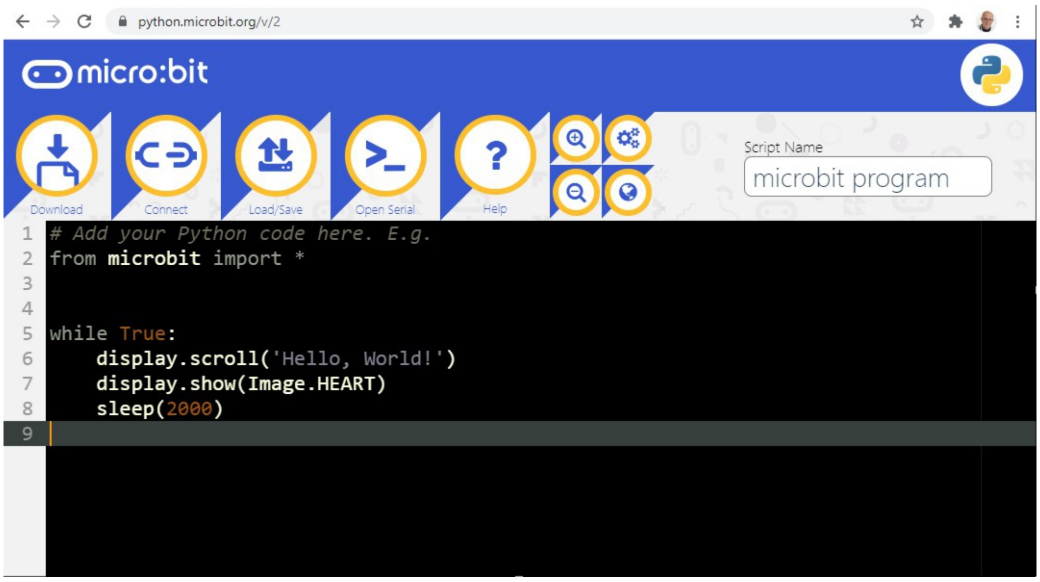


```
from microbit import *
import music
display.off()
sleep(4000)
# -- COMMUNICATION - cannot be changed
spd = 8
def ispis(tekst):
    dd = len(tekst)
    if dd < 31:
        salji(tekst)
    else:
        salji(tekst[0:30]+"+")
        salji(tekst[30:dd])

def salji(tekst):
    global spd
    duz = len(tekst)
    buf = bytearray(duz)
    for n in range(0, duz):
        bb = ord(tekst[n:n+1])
        buf[n] = bb
    try:
        i2c.write(0x11, buf)
        sleep(duz*spd) # ok
    except OSError:
        print("er: salji")

```

Figure 4. MU - MicroPython installed



```
# Add your Python code here. E.g.
from microbit import *


while True:
    display.scroll('Hello, World!')
    display.show(Image.HEART)
    sleep(2000)
```

Figure 5. MicroPython via a web browser

2.2. Basic Modules

2.2.1. Modules for communication between micro:bit and AD interface. (DO NOT CHANGE)

These modules define how data is transferred between micro: bits and the AD BW interface, and should be used **without changing**. If you change some values or parts of the module, **communication may occur** between the micro:bit and the AD interface stops working. **THIS MODULE IS REQUIRED.**

```
pit= bytearray(1)
pit[0] = 1
spd = 8
def ispis(tekst):
    dd = len(tekst)
    if dd < 31:
        salji(tekst)
    else:
        salji(tekst[0:30]+"+")
        salji(tekst[30:dd])

def salji(tekst):
    global spd
    duz = len(tekst)
    buf = bytearray(duz)
    for n in range(0, duz):
        bb = ord(tekst[n:n+1])
        buf[n] = bb
    try:
        i2c.write(0x11, buf)
        sleep(duz*spd)
    except OSError:
        print("er: salji")
```

Figure 6.

2.2.2. **RESET program module** (DO NOT CHANGE)

This module resets (reset) a program running on the AD BW interface. It is **recommended to use** this module (run - rest()) at the beginning of each program, so as not to mix the old ones data stored in the AD interface memory with the new data. **NOT REQUIRED.**

```
def rest():
    global pit
    ispis("RST")
    while True:
        trazi()
        if pit[0] == 5:
            break
        sleep(20)
```

Figure 7.

2.2.3. **Module for reading data from the AD interface** (DO NOT CHANGE)

This module requests a value from the AD interface. Values are used to determine events in the game, such as: winning points, losing a life, or falling. If you do not play with sound or light effects this module is **NOT REQUIRED**.

```
def trazi():
    global pit
    try:
        pit = i2c.read(0x11,1) # AD BW display adresa
    except OSError:
        print("er: trazi")
```

Figure 8.

2.2.4. **Module for sound and light effects** (**CAN BE CHANGED**)

This module is used in game programs. The values tested in the module are specified and **cannot be changed**. Parts that are **GREEN** can be changed. If you want to use this module, you must include the previous module (trazi() - which reads values) in the program. **NOT REQUIRED**.

```
def zvuk():
    global pit
    if pit[0] == 2:  # POINT
        music.pitch(1500, 50)
        ispis("LED;R;30")
    if pit[0] == 3:  # LIFE
        music.pitch(800, 100)
        ispis("LED;G;30")
    if pit[0] == 4:  # FALL
        for freq in range(900, 1200, 30):
            music.pitch(freq, 8)
```

Figure 9.

2.2.5. **Message module at the BEGINNING of the program** (**CAN BE CHANGED**)

This module prints a message at the beginning of the program and can be modified according to your needs.

```
def begin():
    ispis("CLS")
    ispis("START;2;1;2")
    ispis("G A M E;3;3")
    sleep(1000)
    ispis("CLS")
```

Figure 10.

2.2.6. **Message module at the END of the program** (**CAN BE CHANGED**)

This module requests a value from the AD interface. Values are used to display the number of points (Score) at the end of the program. If points are won in the program, only part of the green module can be changed. The rest of the module reads the number of points and displays it on the screen. You can change or supplement the print position variables, name, and end message as desired.

```
def end():
    global pit
    sleep(100)
    trazi()
    ispis("CLS")
    ispis("E N D;2;1;2")
    ispis("G A M E;3;3")
    ispis("Score:"+str(pit[0])+";3;4")
    while True:
        sleep(5000)
```

Figure 11.

3. PROGRAMMING - BASIC FUNCTIONS

3.1. The first program - RST function

We use the function at the beginning of the program, and after the basic modules. Running this function deletes the values in all fields used by the AD interface program. As part of the **rest()** module - 2.2.2 Page 4.

```
ispis("RST")
```

Figure 12. Funkcija RST

3.2. The first program - text printing "HELLO" - TEXT function

In the first program we use the **TEXT** function (Figure 6). The default value for size (s) is **1**, so the function can be called without entering a value.

ispis("TEXT;x;y;s;c") ili **ispis("TEXT;x;y;c")**

Figure 13. TEXT function (text mode)

TEXT = the text to be printed
x = 0-13 text mod
y = 0-5 text mod
s = size 1 - 3 (1)
c = color (B-black, W-white)

At the beginning of each program, it is a good idea to use the **rest()** module, which deletes the values used by the micro:bit AD program in the previous operation of the micro:bit program.

Load the program in micro: bit via the **FLASH** command. The text **HELLO** should be displayed on the screen at the position of the entered values (Figure 14).

```
from microbit import *
import music
#  2.04.2021

# -- COMMUNICATION - start
pit= bytearray(1)
pit[0] = 1
spd = 8
def ispis(tekst):
    dd = len(tekst)
    if dd < 31:
        salji(tekst)
    else:
        salji(tekst[0:30] + "+")
        salji(tekst[30:dd])

def salji(tekst):
    global spd
    duz = len(tekst)
    buf = bytearray(duz)
    for n in range(0, duz):
        bb = ord(tekst[n : n + 1])
        buf[n] = bb
    try:
        i2c.write(0x11, buf)
        sleep(duz * spd)
    except OSError:
        print("er: send")
# -- COMMUNICATION --- end
```

```
# --
def trazi():
    global pit
    try:
        pit = i2c.read(0x11,1)
    except OSError:
        print("er: seek")
# --
def rest():
    global pit
    ispis("RST")
    while True:
        trazi()
        if pit[0] == 5:
            break
        sleep(20)
# --
rest()
while True:
    ispis("HELLO!;1;1;B")
    ispis("HELLO!;2;2;2;B")
```

Figure 14. HELLO text printing program (prog01.py)

The basic part of the program up to WHILE TRUE is the same for all examples that are processed.

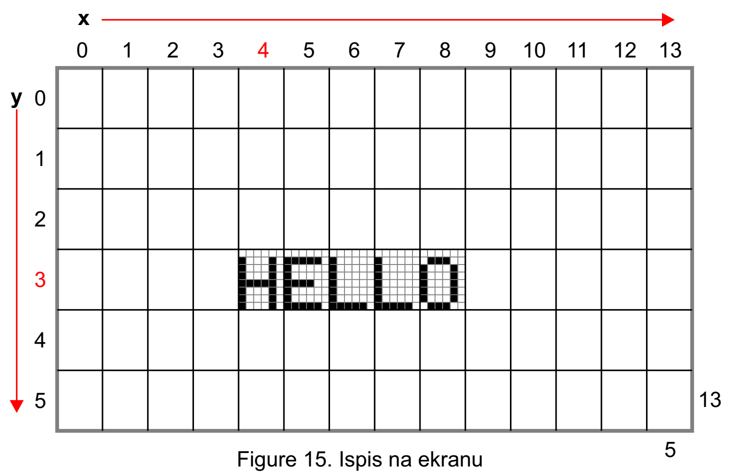


Figure 15. Ispis na ekranu

Print on the screen after starting the first program Figure 15.
By entering a value of 2 or 3 in the **red** field, you can print text larger than the standard dimension (Figure 16).

ispis("TEXT;x;y;s;c")

TEXT = text to print
x = 0-13 text mod
y = 0-5 text mod
s = size 1 - 3 (1)
c = color (B-black, W-white)

Figure 16. Print size

3.3. Multiple text printing via the for function

In this program we use the repeat function - **for** and the variable **n** which we use as the **y** value in the **TEXT** function. Make a change in the previous program according to Figure 17.

```
while True:
    for n in range(0, 6):
        ispis("HELLO!;4;"+str(n)+";B")
```

Figure 17. Multiple text printing (prog02.py)

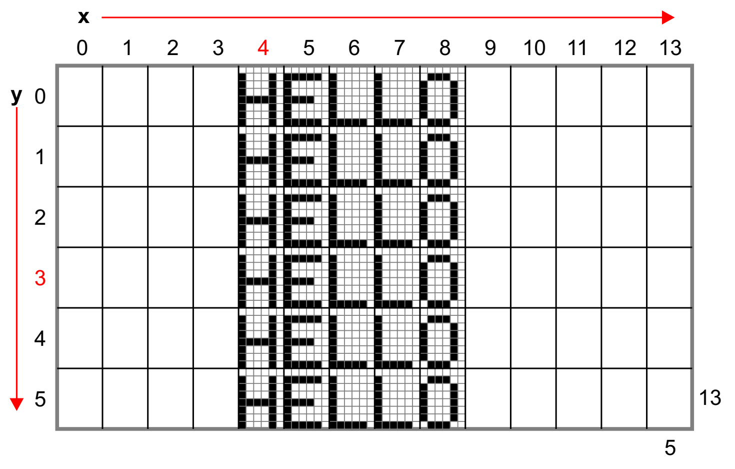


Figure 18. Print on the screen

3.4. Print text in two positions and clear the screen - CLS

In this program we use the **TEXT** function and the **CLS** screen clear function (Figure 19).

```
ispis("CLS")
```

Figure 19. CLS function - clear the screen

We add another **TEXT** function to the program and enter values according to the example. After the first function, we add a half-second pause (500 ms). Repeat this one more time and enter the other values in the new function, as shown in Figure 20.

```
while True:
    ispis("HELLO!;4;1;B")
    sleep(500)
    ispis("HELLO!;4;3;B")
    sleep(500)
```

Figure 20. HELLO text printing program in two positions (prog03.py)

Add a screen clearing function - **CLS** to the program and load the program.

```
while True:
    ispis("HELLO!;4;1;B")
    sleep(500)
    ispis("CLS")
    ispis("HELLO!;4;3;B")
    sleep(500)
    ispis("CLS")
```

Figure 21. HELLO text printing program in two positions (prog04.py)

What is the difference between the screen display?

Remove the **CLS** function from the program. Enter the same values for the position in both **TEXT** functions, and change the print color (B -> W) to the second function.

What is the result of this change?

3.5. Print text in graphics mode (G)

In this program we use the **TEXT** function to print text in graphic resolution (graphic mode Figure 2). With this function we can print text at any position on the screen.

This function does not print text directly on the screen but in the memory (**BUFFER**), so after one or more **TEXT (Graphics)** functions, the **BUF** function must be executed, which displays the recording from the auxiliary memory on the screen. The **default values** are for **s = 1** and **c = B**, so they do not always need to be specified.

ispis("TEXT;x;y;s;c;G") **ili** **ispis("TEXT;x;y;G")**

TEXT = text to print
x = 0-83 graphics mod
y = 0-47 graphics mod
s = size 1 - 3 (**1**)
c = color (B-black, W-white) (**B**)
G = graph. mod (84 x 48)

Figure 22. TEXT function in graphics mode

ispis("BUF")

Figure23. Function for printing auxiliary memory on the screen

```
while True:
    ispis("HELLO!;4;1;B;G")
    ispis("BUF")
```

Figure 24. Example program for printing a single line of text in graphic mode (prog05.py)

```
while True:
    ispis("HELLO!;10;10;G")
    ispis("HELLO!;11;17;G")
    ispis("HELLO!;12;24;1;B;G")
    ispis("HELLO!;13;31;1;B;G")
    ispis("BUF")
```

Figure 25. Example program for printing multiple lines of text in graphics mode (prog06.py)

An example of a program that prints numeric values at a specific position using the TEXT function.

```
while True:
    if button_a.was_pressed():
        ispis("CLS")
        for n in range(1,5):
            ispis(str(n)+". HELLO;10;"+str(n*8)+";B;G")
        ispis("BUF")
```

Figure 26. Example program for printing multiple lines of text in graphics mode (prog07.py)

3.6. Line drawing - LIN

In this program, we use the **LIN** function to draw a line on the screen in graphics mode. For a line, we need to specify the start (x1, y1) and end (x2, y2) point on the screen. We can draw the line in black or white. If we want to delete an already drawn black line, we need to draw a white line in the same position.

Try it!

ispis("LIN;x1;y1;x2;y2;c")

LIN = function name
x1 = 0-83 graphic mode
y1 = 0-47 graphic mode
x2 = 0-83 graphic mode
y2 = 0-47 graphic mode
c = Color (B-black, W-white)

Figure 27. Drawing a line

```
while True:
    if button_a.was_pressed():
        ispis("CLS")
        for n in range(10,40,2):
            ispis("LIN;10;"+str(n)+";60;"+str(n)+";B")
```

Figure 28. Draw more lines (prog08.py)

```
import random - na početku
---

while True:
    if button_a.was_pressed():
        ispis("CLS")
        for n in range(0,30):
            x2 = random.randint(0,83)
            y2 = random.randint(0,47)
            ispis("LIN;0;0;"+str(x2)+";"+str(y2)+";B")
```

Figure 29. Draw multiple lines using the **RANDOM** function (prog09.py)

3.7. Drawing a circle or filled circle - CIR

The **circle** or **filled circle** is not the correct shape for the screen resolution.

In these examples, we use the **CIR** function to draw a circle, in graphical mode. For a circle, we need to determine the position of the center (x, y) of the circle on the screen and the radius. The circle can be drawn in black or white. If we want to delete an already drawn black circle, we need to draw a circle at the same position white. To draw a circle, we use the color **filled with**.

Try it!

ispis("CIR;x;y;r;c;f")

CIR = function name
x = 0-83 graphic mode
y = 0-47 graphic mode
r = radius of the circle (pix)
c = Color (B-black, W-white)
f = Fill color (B, W)

Figure 30. Function for drawing a circle or filled circle

```
while True:
    if button_a.was_pressed():
        ispis("CLS")
        ispis("CIR;40;20;10;B")
        ispis("CIR;40;20;20")
```

Figure 31. (prog10.py)

Draw a circle of radius 10 pixels and radius 20 pixels at position x = 40, y = 20 Figure 31.

```
while True:
    if button_a.was_pressed():
        ispis("CLS")
        ispis("CIR;40;20;10")
        ispis("CIR;40;20;15")
        ispis("CIR;40;20;20")
```

Figure 32. (prog11.py)

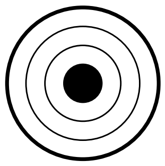
Draw several circles of different radii on the same positionx = 40, y = 20 (Figure 32.).

```
while True:
    if button_a.was_pressed():
        ispis("CLS")
        ispis("CIR;40;20;20;B;B")
        ispis("CIR;40;20;10;B;W")
```

Figure 33. (prog12.py)

Draw a white filled circle of radius 10 inside a black filled circle of radius 20 pixels at position x = 40, y = 20 (Figure 33).

For the exercise, you draw a target with a thick outer edge and a center in black.



3.8. Rectangle drawing - REC

In these examples, we use the **REC** function to draw a rectangle in graphical mode. For rectangles, we need to determine the position of the upper left corner (x, y), width (0-83) and height (0-47). We can draw a rectangle in black or white. If we want to delete an already drawn black rectangle, we need to draw, in the same position, a white rectangle. A rectangle can only be drawn with lines, or **filled with** fill color.

Try it!

ispis("REC;x;y;w;h;c;f")

REC = function name
x = 0-83 graphic mode
y = 0-47 graphic mode
w = width (pix)
h = height (pix)
c = Color (B-black, W-white)
f = Fill color (B, W)

Figure 34. Rectangle drawing function

```
while True:
    if button_a.was_pressed():
        ispis("CLS")
        ispis("REC;10;10;60;30;B")
```

Figure 35. (prog13.py)

Draw a rectangle at position x = 10, y = 10 with a width of 60 and a height of 30 pixels (Figure 35).

```
while True:
    if button_a.was_pressed():
        ispis("CLS")
        ispis("REC;5;5;60;30;B")
        ispis("REC;10;10;50;20;B")
        ispis("REC;15;15;40;10;B")
```

Figure 36. (prog14.py)

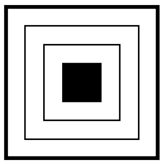
Drawing several rectangles of different positions and sizes (Figure 36).

```
while True:
    if button_a.was_pressed():
        ispis("CLS")
        ispis("REC;5;5;60;30;B;B")
        ispis("REC;15;15;40;10;B;W")
```

Figure 37. (prog15.py)

Draw a white rectangle 40 wide and 10 high, inside a black rectangle 60 wide and 30 pixels high (Figure 37).

For the exercise, you draw a rectangular target with an outer thick edge and a center in black.



3.9. Filling screen - FIL

The **FIL** function fills the screen with bytes of the value entered in the **color** field.
The screen is filled with bytes that are laid vertically as in Figure 38.

ispis("FIL;n")

FIL = function name
n = 0 - 255

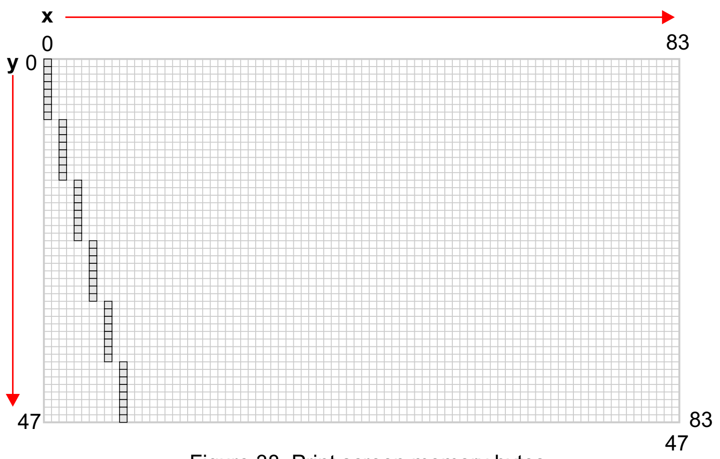


Figure 38. Print screen memory bytes

Example of filling (coloring) the screen with lines spaced one pixel apart. Color value calculation (bytes) you can see in Figure 39.

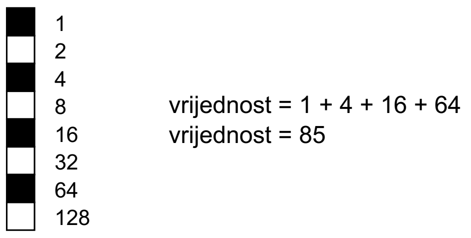


Figure 39. Color calculation (bytes)

```
while True:
    if button_a.was_pressed():
        ispis("CLS")
        ispis("FIL;85")
```

Figure 40. Example program for FIL function (prog16.py)

3.10. Display in black / white or white / black - DIS

The screen can be set to "**normal**" mode (0) - white screen with black print, or in **inverse** (reverse) mode (1) - black screen with white print. By default, the screen is set to "**normal**" mode (0). By change mode the complete screen content is changed via the **DIS** function.
Try the following example in Figure 41. You can supplement it with text.

```
while True:
    if button_a.was_pressed():
        ispis("CLS")
        ispis("DIS;0")
    if button_b.was_pressed():
        ispis("CLS")
        ispis("DIS;1")
```

Figure 41. Example program for DIS function (prog17.py)

3.11. Pixel Print - PIX

ispis("PIX;x;y;c")

PIX	= function name
x	= 0-83 graphic mode
y	= 0-47 graphic mode
c	= Color (B-black, W-white)

Figure 42.

Function for printing pixels on the screen in graphic mode.

```
while True:
    if button_a.was_pressed():
        ispis("CLS")
        ispis("PIX;20;20;B")
```

Figure 43. (prog18.py)

Print one pixel to the ekan in graphic mode, at position x = 20, y = 20.

```
while True:
    ispis("PIX;40;0;B")
    ispis("SBD;2")      # - SBD function page 16
```

Figure 44. (prog19.py)

```
import random

--

while True:
    x = random.randint(0,83)
    ispis("PIX;"+str(x)+";0;B")
    ispis("SBD;1")
```

Figure 45. (prog20.py)

Try the programs in previous figures 44 and 45.

4. PROGRAMMING - SCROLL FUNCTIONS - SCROLL

4.1. Move text UP by one line - SCU

ispis("SCU;r")

SCU = function name
r = rotation (R = yes, "" = no)

Figure 46. SCROLL function to move the text UP

```
ispis("HELLO WORLD;1;5")
while True:
   ispis("SCU;R")
   sleep(500)
```

Figure 47. Example of vertical text shift (prog21.py)

the ext printed at the beginning moves one line up, every half second (Figure 47). Shift text can be used for all print sizes.

The ROTATION function has two states yes = "R" and no = "". Try changing the state to NO (delete; R).
What is the difference in the movement of the text towards the previous state (R)?

4.2. Move text DOWN by one line - SCD

ispis("SCD;r")

SCD = function name
r = rotation (R = yes, "" = no)

Figure 48. SCROLL function of text DOWN

```
ispis("HELLO WORLD;1;0")
while True:
   ispis("SCD;R")
   sleep(500)
```

Figure 49. Example of vertical text shift (prog22.py)

The text printed at the beginning scrolls down one line, every half second (Figure 49). Shift text can be used for all print sizes.

The ROTATION function has two states yes = "R" and no = "". Try changing the state to NO (delete; R).
What is the difference in the movement of the text towards the previous state (R)?

4.3. Move the screen (images) up by one or more pixels (pixel line) - SBU

ispis("SBU;n")

SBU = function name
n = number of pixels

Figure 50. SCROLL function of the screen UP

```
ispis("HELLO WORLD;1;5")
while True:
    ispis("SBU;1")
    sleep(200)
```

Figure 51. Example of moving the screen one pixel UP (prog23.py)

The text printed at the beginning moves one pixel line UP, every 200 milliseconds (Figure 48).
The screen offset (images) can be increased by entering a larger number of pixels.

4.4. Move the screen (images) DOWN by one or more pixels (pixel line) - SBD

ispis("SBD;n")

SBD = function name
n = number of pixels

Figure 52. Funkcija za pomak (SCROLL) ekrana prema DOLJE

```
ispis("HELLO WORLD;1;0")
while True:
    ispis("SBD;1")
    sleep(200)
```

Figure 53. Example of moving the screen one pixel UP (prog24.py)

The text printed at the beginning moves down one pixel line DOWN, every 200 milliseconds (Figure 53).
The screen offset (images) can be increased by entering a larger number of pixels.

4.5. Horizontal screen shift (images) by one pixel - SCC

ispis("SCC;s;a;b;r")

SCC = function name
s = direction of displacement (R, L)
a = from text line (0-5)
b = to text line (0-5)
r = rotation (R)

Figure 54.

Function to move the screen horizontally (images) by one pixel. The function allows you to select the direction of movement (LEFT - left or RIGHT - right), screen areas from line (text) to line or full screen (0-5).

As with text scrolling, the rotation option for circular scrolling of text or image can be turned on.

```
ispis("HELLO WORLD;1;1")
ispis("HELLO WORLD;1;2")
ispis("HELLO WORLD;1;3")
ispis("HELLO WORLD;1;4")
while True:
   ispis("SCC;R;2;3;R")
   sleep(200)
```

Figure 55.

Example of moving the image to the right (Right) of two middle lines of text with a circular display (Figure 55). Add lines to print text in lines 0 and 5, and change the SCC value to 2 in 0 and 3 in 5.

What change happened?

Try changing the direction of the shift.

4.5. Animation with scroll functions

```
from microbit import *
import music
#  2.04.2021

# -- COMMUNICATION - cannot be changed
pit= bytearray(1)
pit[0] = 1
spd = 4
def ispis(tekst):
    dd = len(tekst)
    if dd < 31:
        salji(tekst)
    else:
        salji(tekst[0:30] + "+")
        salji(tekst[30:dd])

def salji(tekst):
    global spd
    duz = len(tekst)
    buf = bytearray(duz)
    for n in range(0, duz):
        bb = ord(tekst[n : n + 1])
        buf[n] = bb
    try:
        i2c.write(0x11, buf)
        sleep(duz * spd)
    except OSError:
        print("er: send")
# -- COMMUNICATION --- end
def trazi():
    global pit
    try:
        pit = i2c.read(0x11,1)
    except OSError:
        print("er: seek")
# --
def rest():
    global pit
    ispis("RST")
    while True:
        trazi()
        if pit[0] == 5:
            break
        sleep(20)
# --
rest()
```

```
ispis("LIN;32;40;52;40")
ispis("LIN;32;41;52;41")
ispis("CIR;20;20;15;B;B")
ispis("CIR;62;20;15;B;B")
for n in range(0,3):
    ispis("CIR;20;20;"+str(n+11)+";W")
    ispis("CIR;62;20;"+str(n+11)+";W")
for n in range(0,4):
    ispis("SCC;L;0;5;R")
    sleep(10)
while True:
    for n in range(0,4):
        ispis("SCC;R;0;5;R")
        sleep(10)
    for n in range(0,4):
        ispis("SCC;L;0;5;R")
        sleep(10)
```

Figure 56. (prog26.py)

Simple animation via various functions.

Try it!

5. PROGRAMMING - GAME FUNCTIONS

5.1. LED light control - LED

The interface has two built-in LED lights, one **RED** and one **GREEN**. RED is below the screen on the left, and GREEN on the right.

ispis("LED;c;t")	LED = function name
	c = color (R - red, G - green)
	t = time in milliseconds

Figure 57.

```
while True:
    ispis("LED;R;100")
    sleep(100)
    ispis("LED;G;100")
    sleep(100)
```

Figure 58.

An example of a program that alternately turns on the RED and GREEN LED lights (Figure 58). (prog27.py)

5.2. Creating BITMAPE (sprite)- objects - BIT

Creating graphic objects (BIT-maps or sprites) is performed in graphic mode (buffer - memory) for faster printing on the screen and avoiding certain bad effects (flickering). Therefore, the first object (one or more objects) are stored in a buffer, and at the end the memory is displayed via the **BUF** function on the screen.

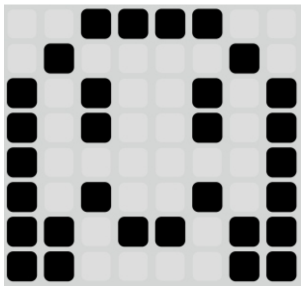
After creating the object (BITMAP), in this example 1, it is necessary to determine the position at which the object will be draw and what COLOR (SPR).
A brief example with basic functions is shown in Figure 60.

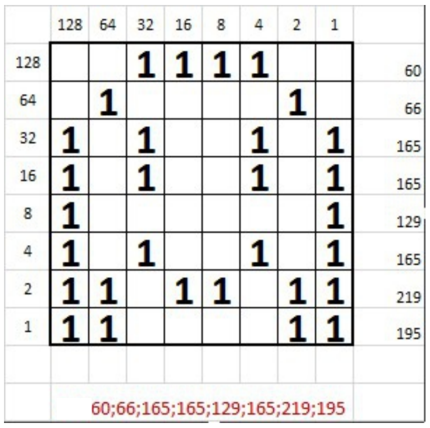
You can create a bitmap in Excel. An Excel example with a template is shown in Figure 59.
When transferring from an Excel file, you need to add a bitmap number (BLACK - 1).
YOU CAN USE ALL BITMAPES AS NEEDED, IF YOU DO NOT USE THE FUNCTIONS WITH WHICH THEY ARE LINKED.

ispis("**BIT;n;b;b;b;b;b;b;b;b**")

BIT = function name
n = bitmap number (1,2,3,4,5,8,9)
b = byte

FIXED BITMAPES: 2 - POINT + 3 - LIFE - 4 - DOOR 8 - ANIMA 9 - PLAYER
FREE BITMAPES: 1 and 5





	128	64	32	16	8	4	2	1	
128			1	1	1	1			60
64		1					1		66
32	1		1			1		1	165
16	1		1			1		1	165
8	1							1	129
4	1		1			1		1	165
2	1	1		1	1		1	1	219
1	1	1					1	1	195

60;66;165;165;129;165;219;195

Figure 59.

without a bitmap number

```
ispis("BIT;1;60;66;165;165;129;165;219;195")
while True:
    if button_a.was_pressed():
        ispis("CLS")
        ispis("SPR;1;40;20;B")   # -- ispis bitmape u memoriji
        ispis("BUF")   # -- ispis memorije na ekran
```

Figure 60. (prog28.py)

The print color allows us to print the object in BLACK and delete it in WHITE.

```
ispis("BIT;1;60;66;165;165;129;165;219;195")
ispis("CLS")
while True:
    for n in range(30,50):
        ispis("SPR;1;"+str(n)+";20;B")
        ispis("BUF")
        ispis("SPR;1;"+str(n)+";20;W")
    for n in range(50,30,-1):
        ispis("SPR;1;"+str(n)+";20;B")
        ispis("BUF")
        ispis("SPR;1;"+str(n)+";20;W")
```

Figure 61. (prog29.py)

Modify the previous program according to Figure 61.

5.5. Displacement controls - horizontal and vertical - BUT

To control the player object in all directions it is necessary to add and control the touch sensors that are located on the underside of the micro:bit. The analog reading of the touch sensors is not the same as the connected USB cable to micro:bit even when not connected. The example in Figure 62 shows the values with USB connected cable. The value without a USB cable connected to the micro:bit is <**100**.

ispis("BUT;s;p;n")

BUT = function name
s = direction + or -
p = X = horizontal Y = vertical
n = number of offset pixels

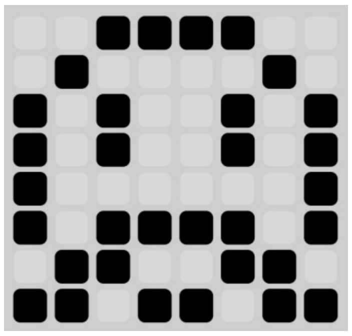
```
while True:
    p1 = pin1.read_analog() # touch sensor
    p2 = pin2.read_analog() # touch sensor
    if button_a.is_pressed():
        ispis("BUT;-;X;1")
    if button_b.is_pressed():
        ispis("BUT;+;X;1")
    if p1 > 240 or p1 < 230:
        ispis("BUT;-;Y;1")
    if p2 > 240 or p2 < 230:
        ispis("BUT;+;Y;1")
```

Figure 62.

In Figure 62, examples can be seen for controlling the horizontal and vertical movement of players by touch sensors or pushbuttons.

5.6. Player object (bitmap 9)

If we want BITMAPA to be a player object, select **9** (**Player**) in the menu. After creating the object it is necessary to run the function to display it on the screen. Text mode resolution is used for positioning (14 x 6). In the following example, the player object is plotted at position x = 5, y = 2.


(9) Player

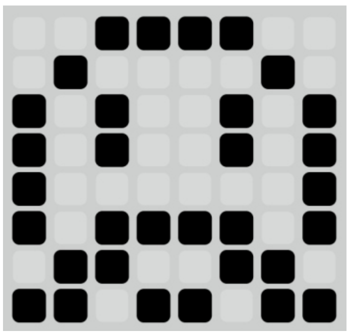
```
ispis("BIT;9;60;66;165;165;129;189;102;219") # player
ispis("POZ;5;2")
while True:
    sleep(5000)
```

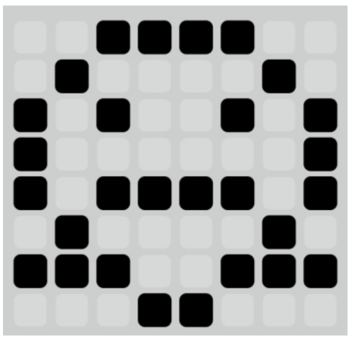
Figure 63. (prog30.py)

The PLAYER function also displays the BUFFER status on the screen, so it is **not necessary** calling the **BUF** function.

5.7. Player object animation (bitmap 8)

Another bitmap is required to animate the player object. Complete the program according to Figure 64. By moving the player objects on the screen are alternately drawn bitmaps **Player** (**9**) and **Player animation frame** (**8**).


(9) Player


(8) Player animation bitmap

```
ispis("BIT;9;60;66;165;165;129;189;102;219") # player
ispis("BIT;8;60;66;165;129;189;66;231;24") # player anima map
ispis("POZ;5;2")
while True:
    p1 = pin1.read_analog() # touch sensor
    p2 = pin2.read_analog() # touch sensor
    if button_a.is_pressed():
        ispis("BUT;-;X;1")
    if button_b.is_pressed():
        ispis("BUT;+;X;1")
    if p1 > 240 or p1 < 230:
        ispis("BUT;-;Y;1")
    if p2 > 240 or p2 < 230:
        ispis("BUT;+;Y;1")
```

Figure 64. (prog31.py)

5.8. Animation speed control - ANI

Animation of a player's object can be faster or slower. To control the speed, we use the **ANI** function (**animation speed**). If we want the animation (bitmap change) to be slower, we need to enter a higher value.

ispis("ANI;n") ANI = function name
n = speed (smaller number higher speed)

```
ispis("BIT;9;60;66;165;165;129;189;102;219") # player
ispis("BIT;8;60;66;165;129;189;66;231;24") # player anima map
ispis("POZ;5;2")
ispis("ANI;1")   #  manji broj veća brzina
while True:
    p1 = pin1.read_analog() # touch sensor
    p2 = pin2.read_analog() # touch sensor
    if button_a.is_pressed():
        ispis("BUT;-;X;1")
    if button_b.is_pressed():
        ispis("BUT;+;X;1")
    if p1 > 240 or p1 < 230:
        ispis("BUT;-;Y;1")
    if p2 > 240 or p2 < 230:
        ispis("BUT;+;Y;1")
```

Figure 65. (prog32.py)

Complete the previous program with the animation speed control function according to Figure 65. Try different speed values.

5.8. START game (2.2.5 - str. 6)

We have the basic construction of the game with the control of the player's object movement. We need to put a message at the beginning which will be printed after the program starts. Insert the function for displaying the standard message **begin()** at the beginning of the program, and after the communication module. After printing, the function must be started sleep, so that the message can be read, and then clear the screen with the **CLS** function. If we do not run the screen clear function, the player object will be drawn over the start text.
This module prints a message at the beginning of the program and can be modified according to your needs.

```
def begin():
    ispis("CLS")
    ispis("START;2;1;2")
    ispis("G A M E;3;3")
    sleep(1000)
    ispis("CLS")  # brisanje ekrana prije nastavka programa
```

Figure 66. (prog33.py)

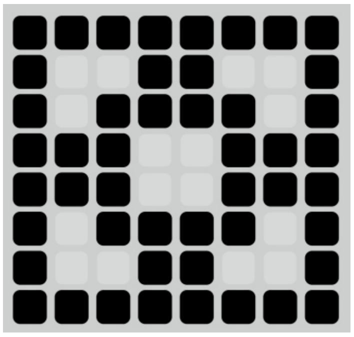
5.9. COLLISION function - KOL

Fill the program with a new object (**1**). Draw it on the screen (**SPR**) several times, on different positions according to Figure 67 or arbitrarily. Complete the program with the **KOL** function. Try which one it is difference with **KOL** function **on** (**1**) and **off** (**0**).

What happens to objects when moving a player's object towards them in both case?

ispis("KOL;s")

KOL	= function name
s	= 0 - no, 1 - yes


bitmapa 1

```
begin()
ispis("BIT;1;255;153;189;231;231;189;153;255")
ispis("BIT;9;60;66;165;165;129;189;102;219") # player
ispis("BIT;8;60;66;165;129;189;66;231;24") # player anima map
ispis("SPR;1;28;28;B")
ispis("SPR;1;36;28;B")
ispis("SPR;1;44;28;B")
ispis("BUF")
ispis("POZ;5;2")
ispis("ANI;1")
ispis("KOL;1")
while True:
    p1 = pin1.read_analog() # touch sensor
    p2 = pin2.read_analog() # touch sensor
    if button_a.is_pressed():
        ispis("BUT;-;X;1")
    if button_b.is_pressed():
        ispis("BUT;+;X;1")
    if p1 > 240 or p1 < 230:
        ispis("BUT;-;Y;1")
    if p2 > 240 or p2 < 230:
        ispis("BUT;+;Y;1")
```

Figure 67. (prog34.py)

5.10. Gravity - GRV

In order for the movement of the player's object to be as natural as possible, and for him to be able to jump and fall, it is necessary to include him in the game gravity function - **GRV** (Figure 68. red).

ispis("GRV;s")

GRV = function name
s = 0 - no, 1 - yes

```
from microbit import *
import music
#  2.04.2021

# -- COMMUNICATION - cannot be changed
pit= bytearray(1)
pit[0] = 1
spd = 8
def ispis(tekst):
    dd = len(tekst)
    if dd < 31:
        salji(tekst)
    else:
        salji(tekst[0:30] + "+")
        salji(tekst[30:dd])

def salji(tekst):
    global spd
    duz = len(tekst)
    buf = bytearray(duz)
    for n in range(0, duz):
        bb = ord(tekst[n : n + 1])
        buf[n] = bb
    try:
        i2c.write(0x11, buf)
        sleep(duz * spd)
    except OSError:
        print("er: send")
# -- COMMUNICATION --- end
def trazi():
    global pit
    try:
        pit = i2c.read(0x11,1)
    except OSError:
        print("er: seek")
# --
def rest():
    global pit
    ispis("RST")
    while True:
        trazi()
        if pit[0] == 5:
            break
        sleep(20)
# --
def begin():
    ispis("CLS")
    ispis("START;2;1;2")
    ispis("G A M E;3;3")
    sleep(1000)
    ispis("CLS")
# --
rest()
begin()
ispis("BIT;1;255;153;189;231;231;189;153;255")
ispis("BIT;9;60;66;165;165;129;189;102;219") # pl
ispis("BIT;8;60;66;165;129;189;66;231;24") # ani
ispis("SPR;1;28;28;B")
ispis("SPR;1;36;28;B")
ispis("SPR;1;44;28;B")
ispis("BUF")
ispis("POZ;5;2")
ispis("ANI;1")
ispis("KOL;1")
ispis("GRV;1")
while True:
    if button_a.is_pressed():
        ispis("BUT;-;X;1")
    if button_b.is_pressed():
        ispis("BUT;+;X;1")
```

To add the GRV function to control the player object we use only horizontal controls. We removed the vertical ones from the program.

The complete program is shown in Figure 67.

What is the difference in the movement of the player's object with on by gravity and off?

Figure 68. Complete program (prog35.py)

5.11. Creating objects (horizontal and vertical) - OBJ - max. 20 objects

Objects longer than one bitmap (8x8 pixels) can be placed horizontally or vertically. Objects are created by repeating the same bitmap several times. **Objects through which points are earned or lost lives** are usually the length of a **one bitmap**, **if they are longer, only the first position is active to obtain points or loss of life.**

ispis("OBJ;e;b;x;y;n;s")

OBJ = function name
e = screen number (1 - 5)
b = bitmap
x = horizontal position (0-10)
y = vertical position (0-5)
n = length (number of repetitions)
s = direction ("" or 0 = hor., 1 = ver.)

```
begin()
ispis("BIT;1;255;153;189;231;231;189;153;255")
ispis("BIT;9;60;66;165;165;129;189;102;219") # player
ispis("BIT;8;60;66;165;129;189;66;231;24") # player anima map
ispis("OBJ;1;1;0;3;6")
ispis("OBJ;1;1;8;0;5;1")
ispis("FX;1")     # -- "screen" display 1
ispis("POZ;5;2")
ispis("ANI;1")
ispis("KOL;1")
ispis("GRV;1")
while True:
    if button_a.is_pressed():
        ispis("BUT;-;X;1")
    if button_b.is_pressed():
        ispis("BUT;+;X;1")
```

Figure 69. (prog36.py)

we made changes to the previous program according to the example in Figure 68. It needs to be defined first objects to be drawn on the screen via the **OBJ** command. All commands are grouped into "**screens**" that are via the **FX** command displayed on the screen. In this example, we define a “**screen**” 1 with two objects. Both objects are composed of the same bitmap (1). For positioning it is necessary to determine the x and y position of the initial object bitmaps. Length (number of repetitions) is determined by entering a value in the length field. Maximum the horizontal length is 11 (84/8 = 10.5 bitmaps).

To read the bitmap vertically, it is necessary to change the value of the "**hor / ver**" field to 1.

```
ispis("OBJ;1;1;0;0;10")
ispis("OBJ;1;1;0;5;10")
ispis("OBJ;1;1;0;1;4;1")
ispis("OBJ;1;1;9;1;4;1")
```

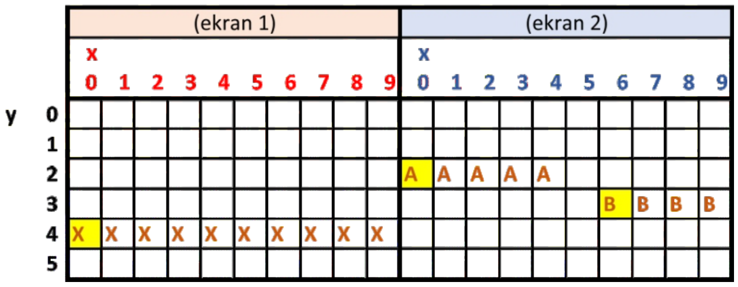
Figure 70. (prog37.py)

Replace the objects defined in the previous example with the objects according to Figure 70.

Don't forget the FX (1) command, which must come from defined objects.

5.12. Creating more than one "screen" - max. 5 "screens"

If we want to create more different "**screens**", it is necessary to create objects for each "**screen**". The following example is with two "**screens**" and three objects.



		(ekran 1)										(ekran 2)									
		x										x									
		0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9
y	0																				
	1																				
	2											A	A	A	A	A					
	3																	B	B	B	B
	4	X	X	X	X	X	X	X	X	X	X										
	5																				

Figure 71.

When you want to create multiple "**screens**" it is good to make a sketch as shown in the example in Figure 71. We can sketch the screens via a table in Excel or raster paper. This makes it much easier to visualize all screens, especially if horizontal and vertical objects are used.

```
begin()
ispis("BIT;1;255;153;189;231;231;189;153;255")
ispis("BIT;9;60;66;165;165;129;189;102;219") # player
ispis("BIT;8;60;66;165;129;189;66;231;24") # player anima map
ispis("OBJ;1;1;0;4;10")
ispis("OBJ;2;1;0;2;5")
ispis("OBJ;2;1;6;3;4;1")
ispis("POZ;5;2")
ispis("ANI;1")
ispis("KOL;1")
ispis("GRV;1")
while True:
    if button_a.is_pressed():
        ispis("CLS")
        ispis("FX;1")
    if button_b.is_pressed():
        ispis("CLS")
        ispis("FX;2")
```

Figure 72. (prog38.py)

The example in Figure 72 uses two “**screens**” that can be alternately displayed via the **A** or **B** key.

5.13. Sliding "screens" - display of screens with horizontal scroll - ASD

When creating a platform game, we use mobile platforms that move from one side of the screen to the other. To run the platforms (objects) we created in the previous program (Figure 74) we need to add activated **ASD horizontal** command (**1**) (Figure 73). Objects on "**screens**" are printed on a circular screen the order of the numbers of the "screen" (1,2,1,2,1,2, ...)

ispis("ASD;s")

ASD = function name
s = ("" or 0) = no, 1 = yes

Figure 73.

The speed of horizontal movement of objects can be changed by the **SPD** command (Figure 74). Lower value variables mean a higher shift rate. The maximum speed is limited to **10**, and the default is set to 100 (0 = 10). We can also increase the speed by shifting by 2 pixels (default 1), but shift more it won't be as 'fine' as a 1 pixel offset. If the program contains a lot of control objects, when the maximum speed (10) may stop working. **In that case you need to reduce the speed because the program does not manage to process all operations in too short a time.**

ispis("SPD;s;p")

SPD = function name
s = speed max.recommended speed = 10
(higher number = lower speed)
p = number of pixels

Figure 74.

5.14. Launch the player object at the sliding screen - jump - JMP

The jump of the 'player' directly upwards is controlled by the JMP command, which is most often used in code platform games where the player is in the same position on the screen (horizontally). The jump height is determined in pixels. If we want, we determine the direction of the jump by entering the + or - variable and the jump angle. To function was active we had to add some more mandatory functions for the position and control of the player's object movement (COLLISION, GRAVITY, PLAYER start position).

ispis("JMP;h;s;a")

JMP = function name
x = jump height - pixels
s = direction (-, +)
a = jump angle (45%) 1 - 5

Figure 75.

```
begin()
ispis("BIT;1;255;153;189;231;231;189;153;255")
ispis("BIT;8;60;66;165;153;66;255;231;0") # player anima
ispis("BIT;9;0;0;60;66;165;153;66;255") # player
ispis("OBJ;1;1;0;4;10")
ispis("OBJ;2;1;0;2;5")
ispis("OBJ;2;1;6;3;3;1")
ispis("FX;1")
ispis("POZ;5;2")
ispis("ANI;5")
ispis("KOL;1")
ispis("GRV;1")
ispis("ASD;1")  # -- auto scroll
ispis("SPD;50;1")  # -- speed 50, offset by 1 pixel
spd = 4  # ---------------- you can increase the speed of communication
while True:
    if button_a.is_pressed():
        ispis("JMP;15")
    if button_b.is_pressed():
        ispis("JMP;15;+;1")
    sleep(100)  # ---- slowing down the excessive use of pushbuttons
```

Figure 76. (prog39.py)

To see what the differences are in the feed rate try a program with different speed values. You can expand the program to another "**screen**" (3).

5.15. Control functions

5.15.1. Game status - GAME status - trazi() (2.2.3 - page 4)

In order for a program running in micro: bit to be able to perform some functions it is necessary read certain values used in the game. Game status is used for performance sound effects during the game and to know when the game is over. The function is called **ONLY ONCE**, before testing the values obtained, or when querying for an X or Y player position.

```
def trazi():
    global pit
    try:
        pit = i2c.read(0x11,1) # AD BW display adresa
    except OSError:
        print("er: trazi")
```

Figure 77.

5.15.2. Sound and light effects - zvuk() (2.2.4 - page 5)

To perform certain sound and light effects associated with a particular event in games (point, loss of life, fall) we use the function **zvuk()**. Mandatory previously run the game status function **trazi()**.

```
def zvuk():
    global pit
    if pit[0] == 2:  # POINT
        music.pitch(1500, 50)
        ispis("LED;R;30")
    if pit[0] == 3:  # LIFE
        music.pitch(800, 100)
        ispis("LED;G;30")
    if pit[0] == 4:  # FALL
        for freq in range(900, 1200, 30):
            music.pitch(freq, 8)
```

Figure 78.

5.15.3. End of game message - end() (2.2.5 - page 5)

In order for the program to end the game with a message, you need to turn on the game end function **end()**. It is mandatory to run the get game status function **trazi()** beforehand.

```
def end():
    global pit
    sleep(100)
    trazi()
    ispis("CLS")
    ispis("E N D;2;1;2")
    ispis("G A M E;3;3")
    ispis("Score:"+str(pit[0])+";3;4")
    while True:
        sleep(5000)
```

Figure 79.

5.15.4. **Points - BOD**

To track the number of points won in the game, it is necessary to run the **BOD** function start with the starting number of points. In a game where we can only win plus points usually the starting value is 0. In a game with the possibility of winning and losing points, the initial value is greater than zero. In order for a **player** to get points, it is necessary to play include **Points (+)** objects.

ispis("BOD;n")

BOD = function name
n = number of points at the beginning of the game (0)

Figure 80.

5.15.5. **Lives - LIV**

To lose a life, in the game, it is necessary to run the **LIV** function, which sets the initial one the value of the number of lives in the game. Loss of life occurs during a fall (**PAD**) or touches the **Lives (-)** object.

ispis("LIV;n")

LIV = function name
n = number of lives at the beginning of the game

Figure 81.

5.15.6. **Fall loss of life - PAD**

The game consists of platforms on which the **player** moves. When falling, the player's object can lose a life or just reappear in the starting position. By turning on the function **PAD** includes loss of life when falling off the screen.

ispis("PAD;b")

LIV = function name
b = 0 - no, 1 - yes

Figure 82.

5.15.7. **Limited game duration - TIM**

To define the duration of the game in seconds, it is necessary to turn on the **TIM** function. We use the function in games where the goal is to collect as many points in equal time limit. In such games, the points-only function is used.

ispis("TIM;t")

TIM = function name
t = time in seconds

Figure 83.

5.15.8. **Negative points - BON**

If you want to turn on the deduction of points, you need to turn on the **BON** function. In order for this function to be active, the **LIV** function must also be activated. This feature takes away points by losing a life.

ispis("BON")

Figure 84.

5.15.9. **The order in which the "screen" is displayed by random selection - RND**

If the game has **MORE THAN TWO** "**screens**", to avoid repeated repetition of the same order we can turn on this feature. The **random** function creates a sequence screen display.

ispis("RND")

Figure 85.

5.16. Complete platform game

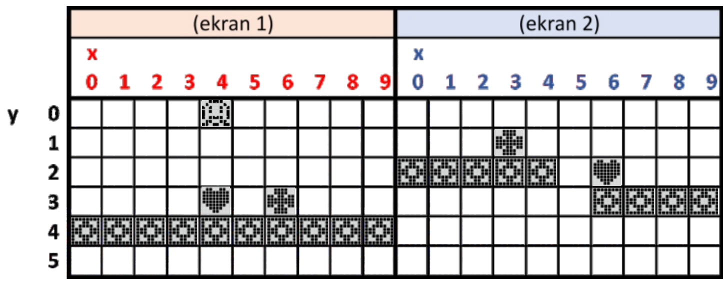


Figure 86.

To make the game complete, we added to each **"screen"** objects for gaining points (♥) and losing lives (✚) according to the positions in the sketch (Figure 86), and the **player** animation object.

```
from microbit import *
import music
#  2.04.2021

# -- COMMUNICATION - cannot be changed
pit= bytearray(1)
pit[0] = 1
spd = 8
def ispis(tekst):
    dd = len(tekst)
    if dd < 31:
        salji(tekst)
    else:
        salji(tekst[0:30] + "+")
        salji(tekst[30:dd])

def salji(tekst):
    global spd
    duz = len(tekst)
    buf = bytearray(duz)
    for n in range(0, duz):
        bb = ord(tekst[n : n + 1])
        buf[n] = bb
    try:
        i2c.write(0x11, buf)
        sleep(duz * spd)
    except OSError:
        print("er: send")
# -- COMMUNICATION --- end
def trazi():
    global pit
    try:
        pit = i2c.read(0x11,1)
    except OSError:
        print("er: seek")
# --
def rest():
    global pit
    ispis("RST")
    while True:
        trazi()
        if pit[0] == 5:
            break
        sleep(20)
# --
```

```
def begin():
    ispis("CLS")
    ispis("START;2;1;2")
    ispis("G A M E;3;3")
    sleep(1000)
    ispis("CLS")
# --
def end():
    global pit
    sleep(100)
    trazi()
    ispis("CLS")
    ispis("E N D;2;1;2")
    ispis("G A M E;3;3")
    ispis("Score:"+str(pit[0])+";3;4")
    while True:
        sleep(5000)
# --
def zvuk():
    global pit
    if pit[0] == 2:  # POINT
        music.pitch(1500, 50)
        ispis("LED;G;30")
    if pit[0] == 3:  # LIFE
        music.pitch(800, 100)
        ispis("LED;R;30")
    if pit[0] == 4:  # FALL
        for freq in range(900, 1200, 30):
            music.pitch(freq, 8)
# --
```

(continuation of the program on the next page)

```
rest()
begin()
ispis("BIT;1;255;153;189;231;231;189;153;255")
ispis("BIT;2;102;255;255;255;255;126;60;24")
ispis("BIT;3;60;60;219;255;255;219;60;60")
ispis("BIT;8;60;66;165;153;66;255;231;0") # player anima
ispis("BIT;9;0;0;60;66;165;153;66;255") # player
ispis("OBJ;1;1;0;4;10")
ispis("OBJ;1;2;4;3;1")
ispis("OBJ;1;3;6;3;1")
ispis("OBJ;2;1;0;2;5")
ispis("OBJ;2;1;6;3;4")
ispis("OBJ;2;2;6;2;1")
ispis("OBJ;2;3;3;1;1")
ispis("FX;1")
ispis("POZ;4;0")
ispis("ANI;3")
ispis("KOL;1")
ispis("GRV;1")
ispis("ASD;1")
ispis("SPD;50;1")
ispis("PAD;1")
ispis("BOD;0")
ispis("LIV;5")
spd = 4
while True:
    if button_a.is_pressed():
        ispis("JMP;15;-;2")
    if button_b.is_pressed():
        ispis("JMP;15;+;2")
    sleep(100)
    trazi()
    if pit[0] != 9:
        zvuk()
    else:
        sleep(300)  # -- time to read end status
        end()
```

Figure 87. (prog40.py)

5.16.1. **Erasing data from memory - DEL**

During program creation and changing object definitions it can happen, that some objects that you delete from the program remain stored in the interface memory. In that In this case, ‘**phantom**’ objects that you have deleted from the program may appear on the screen. To avoid this at the beginning of the program you can turn off and on the **AD interface** or on each time simply add the **DEL** function at the beginning of the program.

This function can be removed from the program after the program is completed.

```
ispis("DEL")
```

Figure 88.

5.16.2. **Automatic level control (levels) of the game - LVL**

To make the game more demanding we can add more weight novelties. The higher the level, the higher the speed of the game and therefore harder to finish. With automatic function control we can determine the values that determine the levels of the game. At the beginning we type a value that defines the maximum speed of the game (last level). After that the initial speed with which we begin the game. By how much the speed of the game increases by moving to a higher level.
The last value determines how many points it takes to win to advance to a higher one level.

This feature may not support all game forms.

Delete the **SPD** function from the previous program and add the **LVL** with the values from examples in Figure 89.

```
ispis("LVL;s;f;d;b")
```

LVL = function name
s = maximum speed (last level)
f = initial speed (first level)
d = increase in speed by the value of d
b = required number of points for a new level

Figure 89. (prog41.py)

5.16.3. **Data exchange rate (micro:bit - AD display) - spd (value)**

At the beginning of the program, when a lot of data is sent to define different functions and objects need to set **spd** to **8** (initial value) or more. That way, the AD interface program has enough time to process all the data. If the time is too short (speed too high), the program will not be able to process everything data sent to it by micro: bit and some objects will be missing or some will not work functions, or the program will stop working.

If you want to take action (after the part of the program that sends the settings for objects and functions) run faster, and to make the game as fast as possible, you can set the **spd** to **4** (lowest recommended value, for highest speed).
A value less than 4 is not recommended.

The AD interface program allows you to try even with smaller values.

```
spd = 8
```

Figure 90.

6. EXAMPLE OF THE PROGRAM

6.1. METEORS

The program includes some of the functions described on page 29. The game has three "**screens**" that are displayed randomly using the **RND** function (5.15.9) and is limited to **30 seconds** by the **TIM** function (5.15.7.). A sketch of the layout of the objects is shown in the figure below (Figure 91).

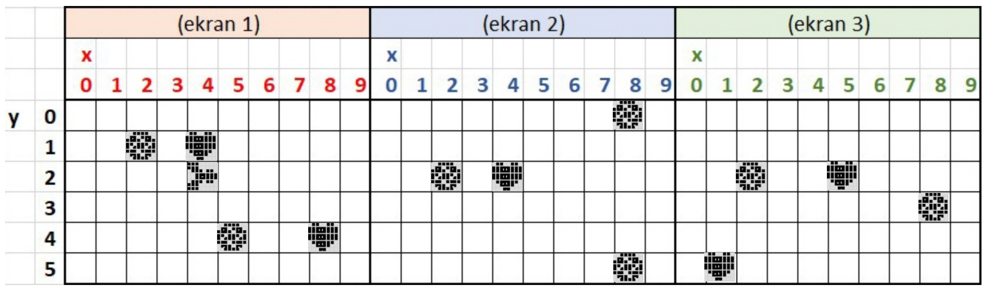


Figure 91.

If you want to make it harder to score points, you can add more objects that will only make it harder for the **player** to move, as well which is shown in Figure 92.

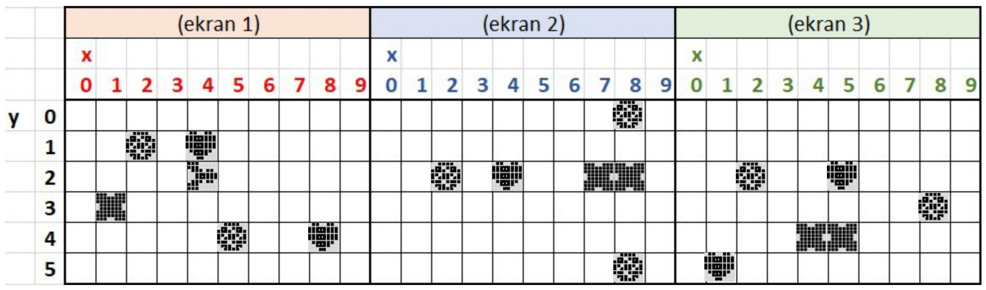
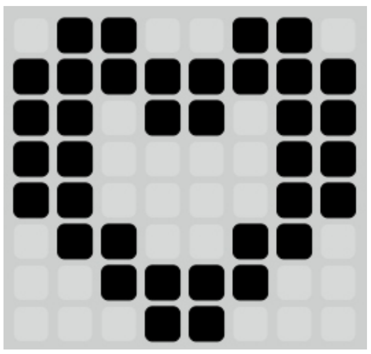
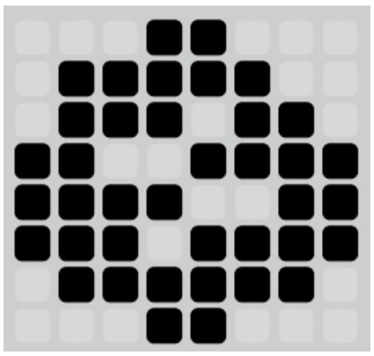


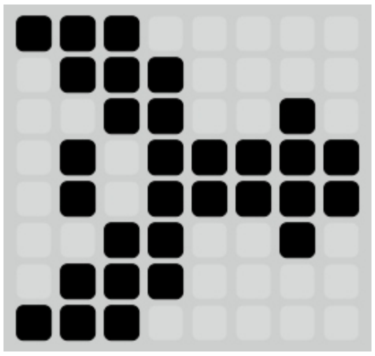
Figure 92.



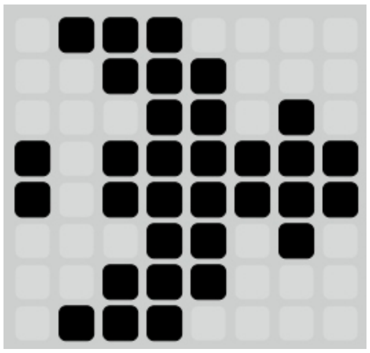
bitmap 2



bitmap 3



bitmap 9



bitmap 8

```
from microbit import *
import music
#  2.04.2021

# -- COMMUNICATION - cannot be changed
pit= bytearray(1)
pit[0] = 1
spd = 8
def ispis(tekst):
   dd = len(tekst)
   if dd < 31:
      salji(tekst)
   else:
      salji(tekst[0:30] + "+")
      salji(tekst[30:dd])

def salji(tekst):
   global spd
   duz = len(tekst)
   buf = bytearray(duz)
   for n in range(0, duz):
      bb = ord(tekst[n : n + 1])
      buf[n] = bb
   try:
      i2c.write(0x11, buf)
      sleep(duz * spd)
   except OSError:
      print("er: send")
# -- COMMUNICATION --- end
def trazi():
   global pit
   try:
      pit = i2c.read(0x11,1)
   except OSError:
      print("er: seek")
# --
def rest():
   global pit
   ispis("RST")
   while True:
      trazi()
      if pit[0] == 5:
         break
      sleep(20)
# --
def begin():
   ispis("CLS")
   ispis("START;2;1;2")
   ispis("G A M E;3;3")
   sleep(1000)
   ispis("CLS")
# --
def end():
   global pit
   sleep(100)
   trazi()
   ispis("CLS")
   ispis("E N D;2;1;2")
   ispis("G A M E;3;3")
   ispis("Score:"+str(pit[0])+";3;4")
   while True:
      sleep(5000)
# --
def zvuk():
   global pit
   if pit[0] == 2:  # POINT
      music.pitch(1500, 50)
      ispis("LED;G;30")
   if pit[0] == 3:  # LIFE
      music.pitch(800, 100)
      ispis("LED;R;30")
   if pit[0] == 4:  # FALL
      for freq in range(900, 1200, 30):
         music.pitch(freq, 8)
# --
rest()
begin()
ispis("BIT;2;102;255;219;195;195;102;60;24")
ispis("BIT;3;24;124;118;207;243;239;126;24")
ispis("BIT;8;224;112;50;95;95;50;112;224") # anim
ispis("BIT;9;112;56;26;191;191;26;56;112") # play.

ispis("OBJ;1;3;2;1;1") -- ekran 1
ispis("OBJ;1;3;5;4;1")
ispis("OBJ;1;2;4;1;1")
ispis("OBJ;1;2;8;4;1")

ispis("OBJ;2;3;2;2;1") -- ekran 2
ispis("OBJ;2;3;8;5;1")
ispis("OBJ;2;2;4;2;1")

ispis("OBJ;3;3;2;2;1") -- ekran 3
ispis("OBJ;3;3;8;3;1")
ispis("OBJ;3;2;1;5;1")
ispis("OBJ;3;2;5;2;1")

ispis("FX;1")
ispis("POZ;4;2")
ispis("ANI;3") -- player animation
ispis("KOL;1")
# -------->              ispis("GRV;1") - we do not use
ispis("ASD;1")
ispis("SPD;30;1") -- brzina pomaka
# -------->              ispis("PAD;1") - we do not use
ispis("BOD;0")
ispis("LIV;5")
ispis("TIM;30")   -- game time 30 seconds
ispis("RND")     -- random order
spd = 4           -- maximum communication speed
while True:
   if button_a.is_pressed():
      ispis("BUT;+;Y;2")
   if button_b.is_pressed():
      ispis("BUT;-;Y;2")
   trazi()
   if pit[0] != 9:
      zvuk()
   else:
      sleep(300)
      end()
```

Figure 93. (prog42.py)

7. EXAMPLE OF THE PROGRAM

7.1. OTHER FUNCTIONS

7.1.1. **Taking the position of a player - GET (X ili Y)**

With this function you can take the value for horizontal (**X**) or vertical (**Y**) the position of the **player** on the screen. Value shows the graphical position of the **player** for x (0-83) or y (0-47) value. You can use this function in any combination when creating program. **The following program will show how to use this function.**

ispis("GET;n")

GET = function name
n = variable (X, Y)

Figure 94.

7.2. Example program

```
from microbit import *
import music
#  2.04.2021

# -- COMMUNICATION - cannot be changed
pit= bytearray(1)
pit[0] = 1
spd = 8
def ispis(tekst):
    dd = len(tekst)
    if dd < 31:
        salji(tekst)
    else:
        salji(tekst[0:30] + "+")
        salji(tekst[30:dd])

def salji(tekst):
    global spd
    duz = len(tekst)
    buf = bytearray(duz)
    for n in range(0, duz):
        bb = ord(tekst[n : n + 1])
        buf[n] = bb
    try:
        i2c.write(0x11, buf)
        sleep(duz * spd)
    except OSError:
        print("er: send")
# -- COMMUNICATION --- end
def trazi():
    global pit
    try:
        pit = i2c.read(0x11,1)
    except OSError:
        print("er: seek")
# --
def rest():
    global pit
    ispis("RST")
    while True:
        trazi()
        if pit[0] == 5:
            break
        sleep(20)
# --
```

```
def begin():
    ispis("CLS")
    ispis("START;2;1;2")
    ispis("G A M E;3;3")
    sleep(1000)
    ispis("CLS")
# --
def end():
    global pit
    sleep(100)
    trazi()
    ispis("CLS")
    ispis("E N D;2;1;2")
    ispis("G A M E;3;3")
    ispis("Score:"+str(pit[0])+";3;4")
    while True:
        sleep(5000)
# --
def zvuk():
    global pit
    if pit[0] == 2:  # POINT
        music.pitch(1500, 50)
        ispis("LED;G;30")
    if pit[0] == 3:  # LIFE
        music.pitch(800, 100)
        ispis("LED;R;30")
    if pit[0] == 4:  # FALL
        for freq in range(900, 1200, 30):
            music.pitch(freq, 8)
# --
```

(continuation of the program on the next page)

```
def dis():                     -- function for reading the x and y positions of players
    global pit
    ispis("   ;0;0;G")         -- delete values
    ispis("   ;65;0;G")
    ispis("GET;X")             -- query for X value
    trazi()                    -- read X value
    ispis(str(pit[0])+";0;0;G")   -- print the X value to the screen
    ispis("GET;Y")             -- query for Y value
    trazi()                    -- read Y value
    ispis(str(pit[0])+";65;0;G") -- print the Y value to the screen
rest()
begin()
ispis("BIT;1;255;153;189;231;231;189;153;255")
ispis("BIT;2;102;255;255;255;255;126;60;24")
ispis("BIT;3;60;60;219;255;255;219;60;60")
ispis("BIT;9;0;0;60;66;165;153;66;255") # player
ispis("OBJ;1;1;0;4;11")
ispis("OBJ;1;3;9;3;1")
ispis("FX;1")
ispis("POZ;3;0")
ispis("KOL;1")
while True:
    p1 = pin1.read_analog() # touch sensor
    p2 = pin2.read_analog() # touch sensor
    if button_a.is_pressed():
        ispis("BUT;-;X;1")
        dis()                 -- calling the dis() function
    if button_b.is_pressed():
        ispis("BUT;+;X;1")
        dis()
    if p1 > 240 or p1 < 230:
        ispis("BUT;-;Y;1")
        dis()
    if p2 > 240 or p2 < 230:
        ispis("BUT;+;Y;1")
        dis()
```

Figure 95. (prog43.py)

In this example, we use the **player** position download functions just to display on the screen. Same functions you can use it to control the **player** or restrict his movement on the screen.

```
def dis():
    global pit
    global x
    global y
  #  ispis("   ;0;0;G")
  #  ispis("   ;65;0;G")
    ispis("GET;X")
    trazi()
    x = pit[0]
  #  ispis(str(x)+";0;0;G")
    ispis("GET;Y")
    trazi()
    y = pit[0]
  #  ispis(str(y)+";65;0;G")
```

```
x = 0
y = 0
while True:
    p1 = pin1.read_analog() # touch sensor
    p2 = pin2.read_analog() # touch sensor
    if button_a.is_pressed():
        if x > 20:
            ispis("BUT;-;X;2")
        dis()
    if button_b.is_pressed():
        if x < 30:
            ispis("BUT;+;X;2")
        dis()
    if p1 > 240 or p1 < 230:
        if y > 8:
            ispis("BUT;-;Y;2")
        dis()
    if p2 > 240 or p2 < 230:
        if y < 16:
            ispis("BUT;+;Y;2")
        dis()
```

(to make the program run faster you can turn it off display values on the screen, according to the above example)

Figure 96. (prog44.py)

Example of limiting the horizontal and vertical movement of players (Figure 95).
In the previous program (Figure 94), make the change according to the example in Figure 95.

8. CONCLUSION

We wanted to create a screen interface that would allow data to be displayed or created simple games. When creating a game, some functions are used to define the operation of the game, a which we also have in real computer games (gravity). To enable maximum creativity **most functions have no limited value**, which means they will happen errors such as program shutdown or printing of incorrect data on the screen.

We wish you a pleasant work.